ระบบเปลือกยูนิกซ์ (UNIX SHELL)

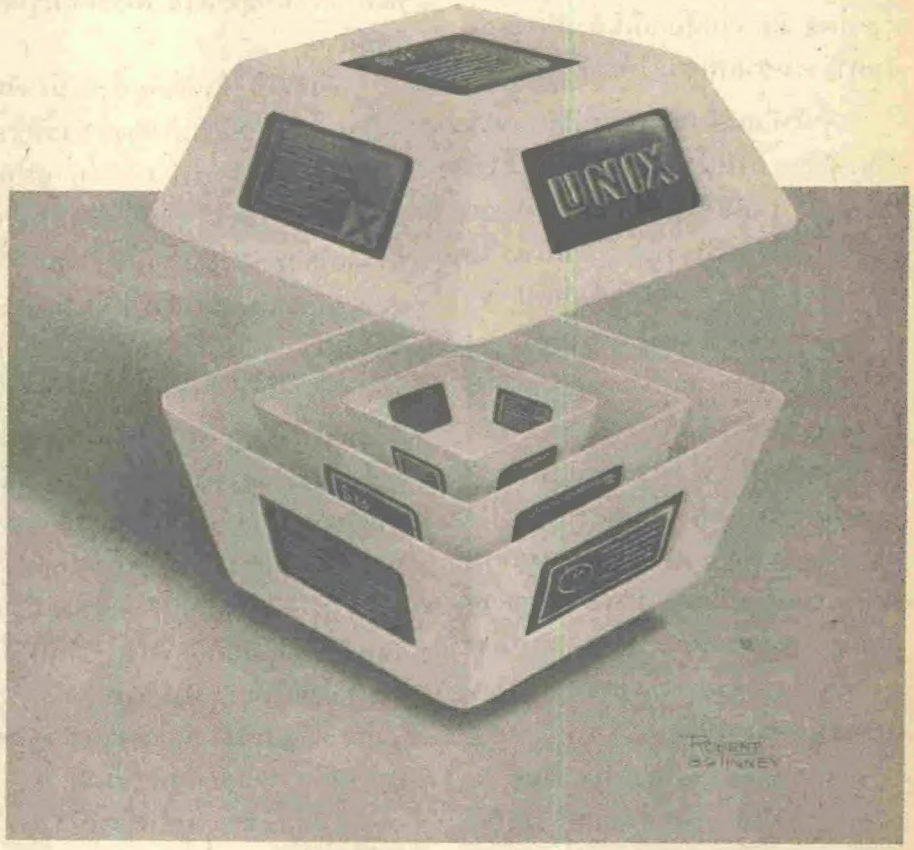
เมื่อผนึก ล่าม + แม่งาน เข้าด้วยกัน

พีรวัฒน์ วัฒนพงษ์

หลายท่านคงจะเคยใช้งานดอสกันมาอย่างโชกโชน ทั้งการใช้งานไฟล์ AUTOEXEC.BAT ทั้งการใช้งานแบตช์ไฟล์อื่น ๆ รวมทั้งการโอนย้ายผลลัพธ์จากจอภาพไปที่ไฟล์ และความสะดวกอื่นที่ดอสมีให้ หลายท่านอาจจะรู้สึกว่านี่แหละสุดยอดแห่งระบบปฏิบัติงาน (OPERATING SYSTEM) แล้วดูกร! สหายร่วมยุทธจักรทั้งหลาย ยังมีระบบปฏิบัติงานอีกอันหนึ่งที่มีโปรแกรมอันสลับซับซ้อนสำหรับช่วยงานให้ท่านได้เหมือนดอสแทบทุกอย่าง ที่จริงมากกว่าอีกหลายเท่าตัวด้วยซ้ำ! สิ่งที่ว่านี้คือ ระบบเปลือกของยูนิกซ์นั่นเอง

ระบบเปลือกของยูนิกซ์ปัจจุบันนั้นมีหลายตัว แต่ที่ดั้งเดิมมาตรฐานที่สุดแล้วรับประกันว่ามีทุกเครื่องที่ให้ยูนิกซ์ คือ โบนเชลล์ (Bourne Shell) สร้างโดย STEPHEN R.BOURNE นอกจากนี้ก็มีซีเชลล์ (C SHELL), คอร์นเชลล์ (KORN SHELL), นิวซีเชลล์ (NEW C SHELL) (บนเครื่อง NEC) และอื่น ๆ อีกมาก แต่หลักการใช้งานใกล้เคียงกันมาก แตกต่างตรงคุณสมบัติพิเศษบางประการเท่านั้น ในบทความนี้จึงกล่าวถึงโบนเชลล์เป็นหลัก

ระบบเปลือกนั้นเป็นส่วนติดต่อกับผู้ใช้ (User Interface) ถ้าเทียบกับดอสแล้วเท่ากับ COMMAND.COM นั่นแหละ แต่ประสิทธิภาพนั้นยอดเยี่ยมสาหัสกว่ากันมาก ไม่เชื่อถาม บิลล์ เกตส์ แห่งไมโครซอฟต์ดูได้ว่างานนี้ใครลอกใครมากันแน่!



ดูกร! สหายร่วมยุทธจักรทั้งหลาย จะขอแนะนำระบบปฏิบัติการอันหนึ่ง ที่มีประสิทธิภาพสูงกว่าดอสหลายเท่านัก สิ่งที่ว่านี้คือ ระบบเปลือกของยูนิกซ์

ระบบเปลือกของยูนิกซ์ (UNIX Shell) นั้นทำงานเป็น 2 ลักษณะด้วยกัน หนึ่งคือ เป็นตัวแปลคำสั่ง ถ้าอธิบายแบบชาวบ้าน ๆ ก็คือทำหน้าที่เป็น "ล่าม" ให้กับผู้ใช้ คือเป็นคนสั่งให้ยูนิกซ์ทำงานต่าง ๆ แทนผู้ใช้ ดังนั้นผู้ใช้ก็จะสั่งการทำงานด้วยภาษาง่าย ๆ เข้าใจสะดวก ตัวเปลือกจะนำคำสั่งเหล่านั้นมาแปล จัดความเรียบร้อยแล้วก็สั่งให้ตัวยูนิกซ์ทำงาน ดังนั้นเมื่อมีเปลือกอยู่ทั้งผู้ใช้ทั้งตัวยูนิกซ์ก็แฮปปี้ต่างคนต่างทำงานร่วมกันได้อย่างมีความสุข โดยมีเปลือกเป็นตัวกลางคอยจัดการงานสัพเพเหระให้

หน้าที่อันดับสองของเปลือกของยูนิกซ์คือ ทำหน้าที่เป็นเปรียบเสมือน "แม่งาน" ให้กับผู้ใช้ ทำให้ผู้ใช้สามารถออกคำสั่งการทำงานติดตามกันรวดเดียวหลาย ๆ คำสั่งที่ได้ เอาผลลัพธ์ของคำสั่งแรกมาเป็นข้อมูลของอีกคำสั่งหนึ่งก็ได้ เปลี่ยนลักษณะผลลัพธ์(Redirect) ก็ได้ ตัดสินใจการทำคำสั่งต่อไป โดยพิจารณาจากผลลัพธ์ของคำสั่งก่อนหน้านั้นก็ได้ กล่าวโดยสรุปก็คือ เปลือกของยูนิกซ์นี้ผู้ใช้สามารถมองเสมือนว่าเป็นภาษาอีกภาษาหนึ่งด้วย ขีดความสามารถสองสถานะ อันนี้เองทำให้ผู้ใช้งานยูนิกซ์มีสไตล์การใช้งานเครื่องไม่เหมือนใคร

เปลือกยูนิกซ์นี้สามารถอ่านคำสั่งเข้ามาจากไฟล์ หรือรับจากเทอร์มินัลก็ได้ คำสั่งต่าง ๆ จะเขียนอยู่ในรูปของ "คำ" ซึ่งคั่นด้วยการเว้นวรรค โดยคำแรกจะเป็นชื่อของคำสั่งที่ต้องการให้ยูนิกซ์ปฏิบัติ คำที่ตามมาคือ ข้อมูลเพิ่มเติมของคำสั่ง ตัวอย่างเช่นคำสั่ง Is-l จะพิมพ์รายชื่อ (LIST) ของไฟล์ให้ไดเรคทอรีปัจจุบันออกมา ข้อมูลเพิ่มเติม -l จะบอกให้คำสั่ง ls รู้ว่าต้องการให้พิมพ์ข้อมูลเกี่ยวกับขนาด สถานะ วัน

ที่แก้ไขครั้งล่าสุดออกมาพร้อมกับรายชื่อไฟล์ด้วย

คำสั่งในยูนิกซ์นั้น ถ้าจะมองไปก็เหมือนการเรียกใช้โปรแกรมย่อย (Procedure) ในภาษาปาสคาลหรือฟอร์แทรนนั่นแหละ จะมีข้อแตกต่างอยู่บ้างก็เพียงสองประการคือ โดยปกติข้อมูลเพิ่มเติมที่จะส่งให้โปรแกรมย่อยจะต้องอยู่ในวงเล็บ สำหรับของเปลือกยูนิกซ์นี้ไม่ต้องใช้วงเล็บ อีกอย่างหนึ่งก็คือการส่งข้อความในภาษาอื่น ๆ มักจะต้องใส่เครื่องหมายคำพูดปิดหัวท้าย สำหรับเปลือกยูนิกซ์นั้นใช้การเว้นวรรคในการตัดข้อความ กรณีนี้สามารถเข้าใจได้ เนื่องจากเปลือกของยูนิกซ์นี้วัตถุประสงค์หลักคือต้องการให้มีใช้ในการสั่งการให้ยูนิกซ์ทำงาน ฉะนั้นจึงควรจะเขียนง่ายปราศจากตัวอักษรประหลาด ๆ และกฎหยุมหยิม

เมื่อจะทำงานคำสั่งหนึ่ง ๆ นั้น เปลือกยูนิกซ์จะสร้างโปรเซสใหม่ขึ้นมาทำงานคำสั่งนั้น โดยเปลือกจะรอจนคำสั่งนั้นทำเสร็จ ซึ่งเป็นหลักการขั้นพื้นฐานของยูนิกซ์ ถ้าต้องการใช้งานเปลือกทำงานอื่นโดยไม่ต้องการรอให้งานเก่าเสร็จก่อน เราก็สามารถทำได้โดยการเพิ่มเติมสัญลักษณ์ต่อท้าย (Postfix) เข้าไปอีกหนึ่งตัวคือ อักษร & ตัวอย่างเช่น print file & นั้น จะสั่งให้ยูนิกซ์พิมพ์ไฟล์ชื่อ file และให้ทำงาน "หลังฉาก" (background) คือให้แยกไปทำงานต่างหากไม่ต้องพัวพันอยู่กับเปลือก เพื่อเราจะได้สามารถใช้เปลือกได้ต่อไป จะเป็นได้ว่าตัวอักษร & นี้เป็นสัญลักษณ์พิเศษของเปลือกโดยเฉพาะ ซึ่งจะถูกดักและตีความหมายโดยตัวเปลือกเอง และจะไม่ถูกส่งให้กับคำสั่ง print ที่เราเรียกใช้

โดยปกติแล้ว ข้อมูลต่าง ๆ ที่จะป้อนให้กับคำสั่งหรือให้แก่เปลือกนั้นจะพิมพ์เข้าจากแป้นพิมพ์ และในขณะเดียวกันผลลัพธ์ก็จะกลับมาปรากฏบนจอภาพ เราสามารถจะโอนย้ายผลลัพธ์โดยเฉพาะเป็นคำสั่ง ๆ ไปยังอุปกรณ์อื่น ๆ หรือนำไปเก็บในไฟล์ได้ โดยการใช้สัญลักษณ์พิเศษ > ดังตัวอย่างเช่น

```
ls -l > file
```

ในส่วน > file นี้จะถูกแปลโดยเปลือก โดยไม่ถูกส่งไปให้คำสั่ง ls รับรู้ด้วย ดังนั้นคำสั่ง ls จะทำงานเหมือนปกติแต่ลบข้อความใน file เก่าออกหมด และถ้าไฟล์ชื่อนี้ไม่มีก็จะสร้างขึ้นมาให้ใหม่ ถ้ากรณีที่ไฟล์เก่ามีอยู่แล้วและเราไม่ต้องการให้ข้อมูลเสียไปก็สามารถนำผลลัพธ์ใหม่มาต่อท้ายได้ โดยใช้สัญลักษณ์ >> เช่น

```
ls -l >> file
```

จะนำผลลัพธ์ของคำสั่ง ls มาต่อท้ายไฟล์ชื่อ file โดยไม่ลบข้อมูลเก่าทิ้ง

แม้กระทั่งข้อมูลเข้าก็สามารถโอนย้ายข้อมูลเข้าจากอุปกรณ์อื่น หรือจากไฟล์ได้เช่นกัน โดยใช้สัญลักษณ์พิเศษ คือ < ตัวอย่างเช่น

```
wc < file
```

จะเป็นการเรียกใช้คำสั่ง we (word count) ซึ่งเป็นคำสั่งนับจำนวนตัวอักษรจำนวนบรรทัด และจำนวนคำที่ป้อนให้กับคำสั่งนี้ เมื่อเปลือกเห็น

สัญลักษณ์ < ก็จะนำไฟล์ชื่อ file มาเป็นข้อมูลให้กับคำสั่ง wc ดังนั้นตามตัวอย่างก็จะหมายความว่าให้นับตัวอักษร จำนวนคำ และจำนวนบรรทัดของไฟล์ file นั่นเอง

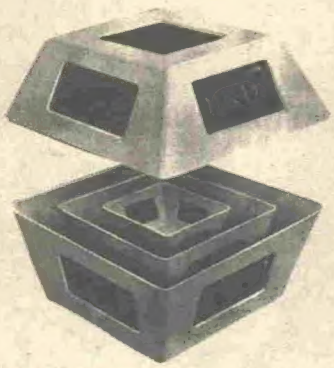
บางครั้งการที่เราจะต้องทำงานหลาย ๆ คำสั่ง แล้วต้องสร้างไฟล์ข้อมูลขึ้นมาคั่นกลาง เพื่อเก็บข้อมูลชั่วคราวไปใช้ก็ทำให้เกิดความยุ่งยากขึ้น ตัวอย่างเช่น ถ้าเราต้องการจะนับในไดเรคทอรีเรามีกี่ไฟล์ (หนึ่งไฟล์ในคำสั่ง ls -l จะพิมพ์หนึ่งบรรทัด) เราก็ต้องใช้คำสั่ง

ls -l > file

นำข้อมูลไดเรคทอรีมาเก็บไว้ในไฟล์ชื่อ temp ไปนับจำนวนบรรทัดก็จะทราบว่ามีไฟล์กี่ไฟล์ ซึ่งเป็นการแบ่งแยกการทำงานเป็นสองท่อน และยังเปลืองที่เก็บ temp อีกต่างหาก (ยังไม่นับการลืมลบไฟล์ temp ทิ้งไป ปล่อยให้อยู่เกะกะอีก) เปลือกยูนิกซ์เล็งเห็นความลำบากลำบนนี้จึงได้จัดให้มีการทำการ "ต่อท่อ" (pipe) ของคำสั่งขึ้น โดยที่ผลลัพธ์ของคำสั่งหนึ่งจะถูกนำไปป้อนเข้ากับอีกคำสั่งหนึ่งโดยตรง จะเห็นได้ว่าการนับจำนวนไฟล์จะลดจากสองขั้นตอนลงมาเหลือขั้นตอนเดียว โดยใช้สัญลักษณ์พิเศษ ดังนั้นตัวอย่างที่กล่าวมาแล้วจะลดรูปเป็น

ls-llwc

ซึ่งให้ผลเช่นเดียวกันแต่ไม่มีไฟล์คั่นกลางให้ยุ่งยาก และก็ไม่ต้องแบ่งขั้นตอนการทำงานด้วย ยูนิกซ์จะทำการ **ยับยั้ง wc ไม่ให้ทำงานจนกว่า ls จะทำ** งานได้ผลลัพธ์ออกมาแล้วชลอ ls ลง เมื่อ wc อ่านข้อมูลไม่ทัน เป็นการจัดความสมดุลย์ให้โดยอัตโนมัติ จะเห็นได้ว่า กรณีนี้ตัวยูนิกซ์โดดเข้ามาจัดการเองไม่ใช่ตัวเปลือกจัดการ



ยูนิกซ์มีคำสั่งหลายตัวที่ใช้เป็น "ตัวกรอง" (FILTER) ซึ่งจะอ่านข้อมูลเข้ามาจัดการดัดแปลงข้อมูลบางสิ่งบางอย่างแล้วพิมพ์ออกมาเป็นผลลัพธ์ ตัวอย่างของตัวกรองก็เช่น คำสั่ง grep (หาข้อความจำเพาะในไฟล์) นั้นจะอ่านข้อมูลทั้งหมดของไฟล์เข้ามาแล้วเลือกแสดงผลลัพธ์เฉพาะบรรทัดที่มีข้อความที่ต้องการเท่านั้น ตัวอย่างเช่น

ls l grep old

จะพิมพ์ชื่อไฟล์ในไดเรคทอรีปัจจุบันที่มีข้อความ old ปรากฏอยู่ในชื่อด้วย

การใช้งานการต่อท่อนั้นไม่ใช่ว่าจะต่อได้ขั้นเดียว เมื่อมาประกอบกับการใช้พวกตัวกรองด้วยแล้วงานของคุณก็จะง่ายเข้าอีกมาก ตัวอย่างเช่น ต้องการนับจำนวนไฟล์ที่มีข้อความ old อยู่ในชื่อไฟล์ในไดเรคทอรีปัจจุบัน ก็สามารถทำได้โดยการใช้คำสั่ง

ls l grep old l wc-1

ก็จะได้ผลลัพธ์ดังต้องการ

นอกจากนี้ เปลือกยังให้โอกาสแก่เราที่จะสร้างข้อความ (โดยเฉพาะชื่อไฟล์) ที่มีลักษณะเป็นกลาง สามารถนำไปเทียบเคียงกับชื่อไฟล์จริงได้ ตัวอย่างเช่น เราเป็นนักเขียนภาษาซีตัวยงเขียนไว้เยอะแยะหลายไฟล์ ทุกไฟล์จะลงท้ายด้วย .C (อยู่แล้ว) ถ้าเราต้องการดูรายชื่อเฉพาะไฟล์ภาษาซีโดยไม่ต้องการดูไฟล์อื่น ๆ ให้ตาลาย เราก็สามารถเรียกใช้สัญลักษณ์พิเศษ * ได้โดยใช้คำสั่ง

IS - 1 *.c

เปลือกจะนำไปทำการ "ขยาย" ให้เป็นการแสดงรายชื่อของไฟล์ "ทั้งหมด" ที่ลงท้ายด้วย .c ออกมาให้ดู ดังนั้นสัญลักษณ์ * จึงมีข้อความที่ประกอบด้วยสัญลักษณ์ตั้งแต่ศูนย์ตัวขึ้นไป นอกจากนี้ยังมีสัญลักษณ์พิเศษอีกหนึ่งตัวคือ? ใช้สำหรับแทนสัญลักษณ์ใด ๆ ก็ได้หนึ่งตัว ในกรณีที่ต้องการที่จะระบุให้แน่ชัดลงไปว่าตัวสัญลักษณ์ที่ต้องการคือตัวอะไร ก็สามารถใช้สัญลักษณ์เพื่อเป็นตัวกำหนดได้ ตัวอย่างเช่น

ls - 1 ? .c

จะแสดงชื่อไฟล์ที่มีชื่อเป็นตัวอักษรเดียวและมีลงท้ายด้วย .c และคำสั่ง

ls [a-z] [0-9] .c

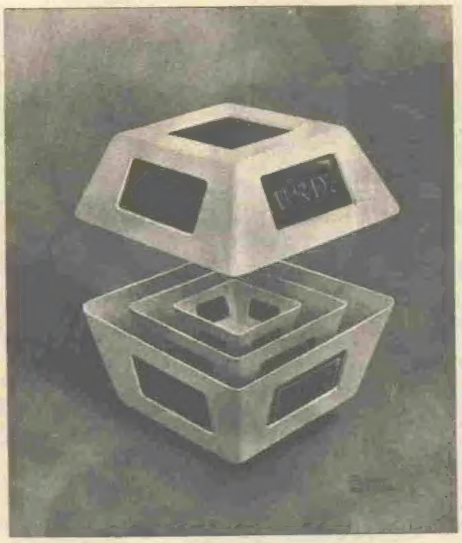
จะแสดงชื่อไฟล์ที่จะมีตัวอักษรตัวแรกจาก a ถึง z และตัวอักษรตัวที่สองจาก 0-9 และมีต่อท้าย .c เท่านั้น บางครั้งการใช้การ "ขยาย" นี้ก็ก่อให้เกิดปัญหาได้เช่นกัน สมมติว่าเราใช้คำสั่ง RM (remove) ซึ่งใช้ลบไฟล์ร่วมกับการใช้สัญลักษณ์พิเศษ * ผลที่ได้คือคำสั่ง RM * จะลบไฟล์ทุกไฟล์ออกจากไดเรคทอรีปัจจุบัน บางครั้งการพิมพ์ผิดพลาดก็เป็นผลให้เกิดโศกนาฏกรรมได้เช่นกัน ตัวอย่างเช่น เราต้องการลบไฟล์ที่ขึ้นต้นด้วย PROJECT เช่น project __ a.c หรือ project __ a.h หรือ project __ b.asm เราก็จะใช้คำสั่ง rm project * โดยมีช่องว่างอยู่หน้า * เปลือกจะแปลความหมายว่าให้ลบไฟล์ชื่อ project จากนั้นก็ลบทุกไฟล์ให้หมด (กัน)

โดยปกติเมื่อเปลือกรับข้อมูลจากจอเทอร์มินัลก็จะมีเครื่องหมายพรอมต์ (PROMPT) ปรากฏขึ้นมา ซึ่งโดยทั่วไปจะให้เป็นเครื่องหมาย $ ถ้าไม่พอใจเราสามารถเปลี่ยนได้เป็นอย่างอื่น โดยการแก้ไขตัวแปร ps1 ดังนี้ ps1 = ready ซึ่งจะเปลี่ยนพรอมต์ไปเป็น ready บางครั้งการใส่ข้อมูลไม่ครบถ้วนก็สามารถก่อให้เกิดพรอมต์ได้เช่นกัน แต่จะเป็นในรูป ซึ่งถ้าเราต้องการเปลี่ยนก็สามารถทำได้ โดยแก้ไขที่ ps2 ดังเช่น ps2=moreinput ก็จะเปลี่ยนพรอมต์ที่ใช้ถามข้อมูลเพิ่มเติมเป็น moreinput

เมื่อเราจะใช้งานยูนิกซ์นั้น ขั้นต้นจะต้องทำการขอใช้งาน (LOGIN) ก่อน ซึ่งเมื่อผ่านขั้นตอนนี้แล้วโดยปกติยูนิกซ์ก็จะจัดการเรียกเปลือกมาทำงานเพื่อรับคำสั่งของผู้ใช้ต่อไป ความไม่สะดวกเกิดขึ้นก็คือบางครั้งเราต้องการให้มีการทำงานบางอย่างทุกครั้งที่เราเริ่มใช้งานยูนิกซ์ก็จะต้องพิมพ์เข้าไปเอง ระบบเปลือกจึงมีการแก้ปัญหานี้โดยการจัดไฟล์พิเศษขึ้นมาหนึ่งชื่อ เรียกว่า profile ซึ่งจะเก็บคำสั่งต่าง ๆ ที่เราต้องการให้ทำงานทุกครั้งที่เริ่มใช้งานเปลือก โดยมาตรฐานแล้ว .profile มักจะมีคำสั่งดังรูปที่ 1

```
date
calendar
mail = /usr/spool/mail/...
term = - - -
export mail term
```

รูปที่ 1



จะเห็นได้ว่าเมื่อเราเริ่มใช้งานเปลือก จะปรากฏวันที่ขึ้นบนจอเนื่องจากคำสั่ง date จากนั้นก็จะมีการตรวจสอบปฏิทินการนัดหมาย โดยคำสั่ง calendar ส่วนตัวแปร mail และ trem ใช้กำหนดที่เก็บจดหมายอิเล็กทรอนิกส์ (ELECTRONIC MAIL) กับใช้กำหนดชนิดของจอเทอร์มินัลตามลำดับ ส่วนคำสั่ง export มีเฉพาะในเปลือกเท่านั้น เป็นการระบุให้ยูนิกซ์ทราบว่า มีการเปลี่ยนแปลงค่าตัวแปรดังกล่าว

ไม่ใช่ว่าเปลือกนั้นจะต้องสั่งการจากจอเทอร์มินัลอย่างเดียว เท่านั้น เราสามารถจะให้เปลือกอ่านคำสั่งที่เราพิมพ์เก็บไว้ก่อนในไฟล์ออกมาทำงานเป็นลำดับได้ ตัวอย่างเช่น คำสั่ง

sh test arg1 arg2 ...

คำสั่ง sh จะเป็นการเรียกใช้เปลือกเพิ่มขึ้นอีกหนึ่งชั้น (ด้วยเหตุนี้จึงเป็นที่มาของชื่อเปลือกหรือเชลล์เพราะเรียกซ้อนทับกันได้หลายชั้นนี่เอง) ส่วน test นั้นคือชื่อไฟล์ที่เราเขียนคำสั่งเก็บไว้ก่อนล่วงหน้าแล้ว และ arg1 arg2... เป็นข้อมูลเพิ่มเติมที่สามารถให้แก่เปลือกได้อีก นอกจากจะได้ใส่ไว้ในไฟล์แล้ว เปลือกเองเอื้อเฟื้อสัญลักษณ์พิเศษมาให้อีกหนึ่งชุด คือ $1 $2...ซึ่งเมื่อใช้ในไฟล์คำสั่งเปลือก (Shell Procedure หรือ Shell Script) จะหมายความถึงตัวข้อมูลเพิ่มเติมซึ่งคิดตามตำแหน่ง อย่างเช่น ไฟล์หนึ่งชื่อ myway มีข้อความ who grep $1 เก็บอยู่ ถ้ามีการเรียกใช้ด้วยคำสั่ง sh myway tony จะทำให้มีความหมายจริง ๆ เทียบเท่ากับคำสั่ง who grep tony ซึ่งเป็นการตรวจสอบรายชื่อผู้ใช้งานเครื่อง ในปัจจุบัน (who) เพื่อหาผู้ใช้ที่ชื่อ tony

บางครั้งเราต้องการทำงานบางคำสั่งซ้ำ ๆ ซาก ๆ เพียงแค่เปลี่ยนข้อมูลไปเรื่อย ๆ เช่น ค้นหาเบอร์โทรศัพท์ของคนรู้จักหลาย ๆ คน เราสามารถจะอาศัยขีดความสามารถของเปลือกช่วยงานให้ง่ายยิ่งขึ้นได้ กรณีเบอร์โทรศัพท์ตามตัวอย่างที่กล่าวข้างต้นสามารถทำได้โดยการเก็บชื่อและเบอร์โทรศัพท์ไว้ในไฟล์ชื่อ telnos โดยจัดข้อมูลในลักษณะดังรูปที่ 2

```
AREE      278-1716
SAKCHAI   579-1111
SOMBOON   231-4652
SOMBOON   2314677
MONGKOL   442-5525
   .          .
   .          .
   .          .
```

รูปที่ 2

ในส่วนของไฟล์คำสั่งเปลือก (ตั้งชื่อว่า tel) เราจะใช้ขีดความสามารถอันหนึ่งของเปลือก คือการทำงานวนซ้ำ โดยใช้คำสั่ง for สำหรับตัวอย่างนี้จะเขียนในลักษณะดังรูปที่ 3

```
for i
do grep $1 < telnos;
done
```

รูปที่ 3

เมื่อเราสั่ง tel AREE SOMBOON จะได้ผลลัพธ์มาเป็นดังรูปที่ 4

```
AREE        278-1716
SOMBOON     231-4652
SOMBOON     231-4677
```

รูปที่ 4

พูดง่าย ๆ ก็คือคำสั่ง for จะเอาชื่อ AREE และ SOMBOON ทีละชื่อไปค้นหาในไฟล์ชื่อ telnos ว่ามีบรรทัดไหนมีชื่อนี้อยู่ก็จะพิมพ์ออกมาโดยพ่วงเบอร์โทรศัพท์มาด้วยในตัว ลักษณะคำสั่งของ for จะจัดอยู่ในรูปที่ 5

ในลักษณะแรกเปลือกจะนำข้อมูลเพิ่มเติม (ARGUMENT) มาแทนใน i ทีละตัวหลังจากนั้นจะทำคำสั่ง 1, คำสั่ง 2,...ไปจนจบถึง done เป็นอันเสร็จพิธี ซึ่งเป็นแบบที่เราใช้ทำระบบค้นเบอร์โทรศัพท์ที่กล่าวข้างต้น ส่วน for ลักษณะที่สอง จะเอา คำ 1, คำ 2, ...มาแทนใน i ทีละตัวตามลำดับแล้วทำงานคำสั่งไปจนถึง done เป็นอันเสร็จเป็นชุด ๆ ไป เช่นกัน

```
foi i                          for i in คำ1 คำ2 คำ3
do คำสั่ง 1;                   do คำสั่ง 1;
   คำสั่ง 2;                หรือ   คำสั่ง 2;
   คำสั่ง 3;                      คำสั่ง 3;
     .                              .
     .                              .
     .                              .
done                           done
```

รูปที่ 5

บางครั้งเราต้องการตัดสินใจทำงานต่าง ๆ กันโดยขึ้นอยู่กับลักษณะของข้อมูลแทนที่เราจะต้องมานั่งพิจารณาข้อมูลก่อนแล้วจึงตัดสินใจเลือกใช้คำสั่งต่าง ๆ เราสามารถจะให้เปลือกเป็นคนช่วยจัดการพิจารณาให้เสร็จสรรพได้ โดยใช้คำสั่ง case ซึ่งเป็นลักษณะการตัดสินใจเลือกได้หลายทาง ลักษณะของคำสั่ง จะเป็นในรูปที่ 6

```
case ตัวแปร in
    รูปแบบ 1)          ชุดคำสั่ง ; ;
    รูปแบบ 2)          ชุดคำสั่ง ; ;
esac
```

รูปที่ 6

เปลือกจะพิจารณาข้อมูลในตัวแปร ถ้าตรงกับรูปแบบใดในทั้งหมดที่มีอยู่ก็จะทำงานชุดคำสั่งนั้น (เฉพาะรูปแบบแรกที่ตรงเท่านั้น) ดังนั้นเราสามารถมีชุดคำสั่งกันเหนียวในกรณีที่ข้อมูล **ไม่ตรงกับรูปแบบที่ใส่ไว้เลย โดยการ** ใช้เครื่องหมาย * ซึ่งได้กล่าวมาแล้วข้างต้น ทำให้กลายเป็นดังรูปที่ 7

คำสั่ง esac ถ้าพิจารณาดูจะเห็นว่าคือการเขียนคำว่า case กลับข้างนั้นเอง เป็นการบอกเปลือกให้รู้ว่า รูปแบบที่จะต้องพิจารณาสิ้นสุดเพียงแค่นี้

ในกรณีเราต้องการจำเพาะเจาะจงที่จะให้เปลือกทำงานชุดคำสั่งหนึ่งเมื่อมีข้อมูลพิเศษเฉพาะ หรือสถานการณ์เฉพาะปรากฏขึ้น เราก็สามารถทำได้โดยใช้คำสั่ง if ซึ่งจะใช้งานในรูปแบบดังรูปที่ 8

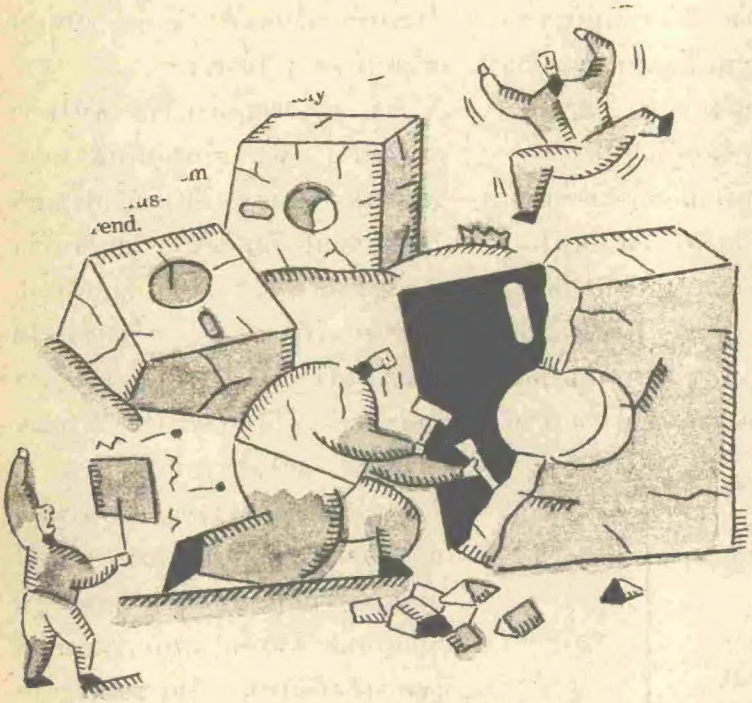
```
case ตัวแปร in
    รูปแบบ 1)          ชุดคำสั่ง 1 ; ;
    รูปแบบ 2)          ชุดคำสั่ง 2 ; ;
    รูปแบบ 3)          ชุดคำสั่ง 3 ; ;
    (*) ชุดคำสั่งกันเหนียว ; ;
esac
```

รูปที่ 7

โดยที่เปลือกจะพิจารณาจากชุดคำสั่ง 1 ว่าได้ผลลัพธ์การทำงานเป็นจริงหรือไม่ (ทำงานสำเร็จหรือไม่) ถ้า

```
if          ชุดคำสั่ง 1
then        ชุดคำสั่ง 2
fi
```

รูปที่ 8



```
echo   "NOW YOU ARE AT pwd"
```

รูปที่ 9

```
NOW YOU ARE AT /usr/ SOMCHAI/src
```

รูปที่ 10

เป็นจริง ก็จะทำงานชุดคำสั่ง 2 ต่อไป แต่ถ้าเป็นเท็จ (ทำงานชุดคำสั่ง 1 ไม่สำเร็จ) ก็จะข้ามไปไม่ทำงานชุดคำสั่ง 2 ในกรณีที่ชุดคำสั่ง 1 มีคำสั่งมากกว่าหนึ่งคำสั่ง ก็จำเป็นจะต้องมีการผสมผลลัพธ์ในเชิงตรรก (จำพวก AND และ OR) ซึ่งเปลือกก็มีสัญลักษณ์ && และ || ใช้แทน AND และ OR ตามลำดับ

เพื่อให้ระบบเปลือกสมบูรณ์ จึงมีการใช้งานตัวแปรในคำสั่งที่ใช้กับเปลือกได้ โดยตัวแปรเหล่านี้จะใช้เก็บข้อมูลเป็นรูปของข้อความ และจะจัดเก็บอยู่กับตัวเปลือกเอง สามารถเรียกมาใช้แก้ไขดัดแปลงได้ ตัวอย่างการกำหนดค่าของตัวแปรจะเป็นในรูป ชื่อตัวแปร = ข้อความที่จะเก็บ เช่น ถ้าเราต้องการเก็บชื่อของเรา (สมมติว่าชื่อ SOMCHAI) ลงในตัวแปรชื่อ username ก็สามารถทำได้โดยเขียนคำสั่ง username = SOMCHAI ข้อความ SOMCHAI ก็จะถูกเก็บในตัวแปรชื่อ username เมื่อเราต้องการได้ข้อมูลที่เก็บในตัวแปรออกมาก็ใช้เครื่องหมาย $ นำหน้าชื่อตัวแปร เช่น $ username จะมีค่าเท่ากับ SOMCHAI การเรียกค่าของข้อมูลอาจทำได้อีกวิธีหนึ่งคือ เขียนในรูป $ {ชื่อตัวแปร} ซึ่งให้ผลเหมือนกันทุกประการ เพียงแต่กรณีหลังนี้ใช้ได้ผลดีเมื่อมีการใช้ตัวอักษรตามมาติด ๆ หลังชื่อตัวแปร วงเล็บปีกกาจะเป็นตัวแยกเปลือกให้ทราบว่าส่วนใดเป็นชื่อตัวแปร ส่วนใดเป็นตัวอักษรจริง ๆ ตัวอย่าง เช่น คำสั่ง

```
echo hello $ {username} SAN
```

จะเป็นการนำข้อความมาพิมพ์บนจอ (echo) โดยจะได้ผลลัพธ์ปรากฏบนจอเป็น

HELLO SOMCHAISAN

ซึ่งมองดูน่ารัก (เป็นญี่ปุ่นดี!)

นอกจากนี้แล้วผลลัพธ์ที่ได้จากการทำคำสั่งยังสามารถจะนำมาใส่ลงในข้อความต่าง ๆ ได้ โดยใช้สัญลักษณ์ “คำสั่ง” ยกตัวอย่างดังรูปที่ 9

ซึ่งเป็นการนำเอาบรรทัดที่คำสั่ง echo พิมพ์ออกไปนั้น จะมีการแทนคำสั่ง pwd (ที่อยู่ปัจจุบัน) ด้วย / user /SOMCHAI/src ซึ่งเป็นไดเรคทอรีปัจจุบัน(สมมติ)ที่เราทำงานอยู่ ผลลัพธ์ของคำสั่ง echo จึงได้เป็นดังรูปที่ 10

มาถึงตรงนี้ก็นับว่าได้ทำความรู้จักกับระบบเปลือกของยูนิกซ์มาบ้างแล้วพอสมควร (แต่รับประกันว่ายังมีอีกมาก!) ท่านผู้อ่านทั้งหลายก็คงจะสามารถนำเอาเปลือกของยูนิกซ์นี้มาใช้งานให้เกิดประโยชน์ เป็นเครื่องมือลดภาระการที่จะต้องพิมพ์คำสั่งมาก ๆ ลงไปได้บ้าง ซึ่งข้อมูลเท่าที่ปรากฏในบทความนี้ก็พอสำหรับการใช้งานขั้นต้นแล้ว ส่วนถ้าต้องการใช้งานระดับสูงขึ้นกว่านี้ ขอให้อดใจรอภาคสองไปก่อน (ยังไม่มีแรงเขียน) หรือถ้ารอไม่ไหวจะหาหนังสือภาษาอังกฤษอ่านเลยก็ได้ เล่มหนึ่งที่น่าอ่านอย่างยิ่งคือ "SHELL PPOGRAMMING" ของ STEPHEN R. BOURNE บุรุษผู้ให้กำเนิดโบนเชลล์นี้เอง ขอแนะนำว่าสำหรับผู้ใช้งานยูนิกซ์ถ้าต้องการก้าวหน้าต่อไป การเข้าใจและใช้ประโยชน์ระบบเปลือกให้กล่องแกล่วมีความสำคัญไม่น้อย เมื่อคุณใช้จนติดใจแล้วจะพบว่าเหมือนรถยนต์ติดเทอร์โบอย่างไรอย่างนั้นเชียว!